MC202—Estruturas de Dados
Lista de Exercícios 1

Islene Calciolari Garcia

1 Listas Ligadas

```
typedef struct no {
   int v;
   struct no* prox;
} No;
```

1.1 Escreva uma função recursiva e outra não recursiva para contar o número de elementos na lista ligada apontada por `p`:

```
int nElementos(No* p);
```

1.2 Escreva uma função recursiva e outra não recursiva que inverte a lista ligada apontada por `p`. Nenhum nó auxiliar deve ser criado.

```
No* inverteLista(No* p);
```

1.3 Considere que os elementos da lista ligada estão ordenados. Escreva uma função que remove um elemento com valor v da lista l. Caso nenhum elemento com este valor seja encontrado, a função deve retornar 0.

```
int remove(No** l, int v);
```

Por que é necessário utilizar um parâmetro do tipo `No**`? Se a lista tivesse um nó cabeça, um parâmetro do tipo `No*` seria adequado?

Considerando a declaração fornecida (com `No**`) você consegue escrever uma versão recursiva desta rotina?

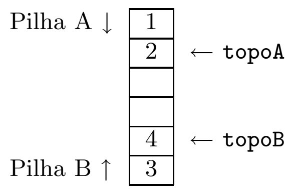
1.4 Escreva uma função que recebe duas listas `x`$= (x_1, \ldots, x_n)$ e `y`$= (y_1, \ldots, y_m)$ como parâmetro e retorna uma lista formada pelos elementos de x e y intercalados. A lista resultante será da forma $(x_1, y_1, \ldots, x_m, y_m, x_{m+1}, \ldots, x_n)$ se $m \leq n$ ou $(x_1, y_1, \ldots, x_n, y_n, y_{n+1}, \ldots, y_m)$ se $m > n$. Nenhum nó adicional deve ser criado e as listas `x` e `y` devem ficar vazias ao final (receber NULL).

```
No* intercalaListas(No** x, No** y);
```

Escreva uma versão recursiva e outra não recursiva.

2 Pilhas

2.1 Duas pilhas A e B podem compartilhar o mesmo vetor, como esquematizado na figura a seguir. Faça as declarações de constantes e tipos necessárias e escreva as seguintes rotinas: (i) o procedimento `criaPilhas` que inicia os valores de topoA e topoB; (ii) as funções `vaziaA` e `vaziaB`; (iii) os procedimentos `empilhaA`, `empilhaB`, `desempilhaA` e `desempilhaB`. Só deve ser emitida uma mensagem de pilha cheia se todas as posições do vetor estiverem ocupadas.



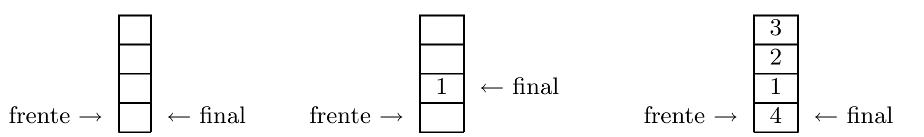
2.2 Escreva uma função que verifique se uma *string* de entrada é da forma xCy, tal que x é uma *string* composta por caracteres `A` e `B` e y é a *string* reversa de x. Por exemplo, a cadeia `ABABBACABBABA` é do formato especificado. A *string* de entrada deve ser lida seqüencialmente.

```
int xCy(char* str);
```

2.3 Escreva um procedimento que verifique se uma *string* de entrada formada apenas por `'('` e `')'` está balanceada. É necessária a utilização de uma pilha para resolver este problema?

3 Filas

3.1 Considere uma implementação de fila utilizando vetor circular, que utiliza apontadores (índices) para a frente e o final da fila: `frente` aponta para a posição imediatamente anterior ao primeiro elemento da fila e `final` aponta para o último elemento inserido, se existir. Comente a dificuldade para se diferenciar fila cheia de fila vazia.



(a) Fila vazia (b) Fila com um elemento (c) Fila com quatro elementos

Implemente as funções `cria_fila`, `fila_vazia`, `insere_file` e `remove_fila` utilizando um *flag* (variável booleana) para indicar se a fila está cheia ou vazia.

3.2 Uma fila *simétrica*[1] permite inserção e remoção em ambas as extremidades. Faça as declarações de tipo apropriadas e escreva as seguintes rotinas para implementação utilizando lista ligada de filas simétricas: `cria_fila`, `fila_vazia`, `insere_frente`, `insere_final`, `remove_frente` e `remove_final`.

Considere a seguinte restrição adicional: todas estas operações devem ser implementadas em tempo constante, ou seja, nenhuma precisa percorrer todos os elementos da fila para ser executada. Você consegue escrever uma implementação que respeita esta restrição utilizando listas ligadas simples? E utilizando listas duplamente ligadas?

3.3 Podemos aproveitar uma implementação para fila simétrica para implementar uma fila ou uma pilha. Mostre como isso poderia ser feito.

[1] Em inglês, *doubled-ended queue (deque).*

4 Listas Generalizadas

```
enum elem_t {tipo_int, tipo_char, tipo_sublista};

nion info_lista {
  int i;
  char c;
  struct No* sublista;
};

typedef struct No {
  enum elem_t tipo;
  union info_lista info;
  struct No* prox;
} No;
```

4.1 Escreva uma função recursiva que retorna a soma de todos os elementos inteiros em uma lista generalizada ou 0, caso a lista esteja vazia.

```
int soma(No* l);
```

4.2 Escreva uma função recursiva que retorna 1 caso exista algum número negativo entre os inteiros presentes na lista generalizada ou 0 caso contrário. Você não deve visitar nós desnecessariamente caso tenha encontrado um número negativo.

```
int existe_negativo(No* l);
```

4.3 Escreva uma função que retorna 1 caso as duas listas passadas como parâmetro sejam iguais e 0 caso contrário.

```
int iguais(No* l1, No* l2);
```

4.4 Escreva uma função que inverte o conteúdo de uma lista ligada generalizada. Por exemplo, o inverso da lista (((1,2),3,4),(5,6),7,(8)) é ((8),7,(6,5),(4,3,(2,1))).

```
No* inverte(No* l);
```

4.5 Escreva uma função que verifica se uma lista s está contida em uma lista l. Por exemplo, (2,3) está contida em (1,2,3) e (3,(4,5)) está contida em (1,(2),(3,(4,5)),6). Uma lista vazia está contida em qualquer lista.

```
int contida(No* s, No * l);
```

4.6 Considere a definição abaixo para a profundidade de listas generalizadas e escreva uma função que retorna a profundidade de uma lista l.

$$prof(l) = \begin{cases} 0 & \text{se } l \text{ é um átomo} \\ 1 + max\{prof(x_1), \ldots, prof(x_n)\} & \text{se } l \text{ é a lista}(x_1, \ldots, x_n) \end{cases}$$

```
int prof(No* l);
```

4.7 Suponha que um programador, para tentar aumentar a legibilidade das listas impressas resolveu adotar a seguinte política: as listas de nível mais baixo são delimitadas por (), as de nível imediatamente mais alto por [] e as outras por {}.

Só com ()	Com (), [] e {}
()	()
((2,3),5)	[(2,3),5]
((((1),2),3),(((4))))	{{[(1),2],3},{[(4)]}}

Seguindo esta idéia, implemente a função abaixo. Defina e implemente funções auxiliares, caso ache necessário.

```
void imprime(No* l);
```

5 Apontadores e Linguagem C

5.1 O objetivo do programa abaixo é (i) iniciar uma lista ligada, indicando que esta está vazia, (ii) inserir um elemento no início desta lista e (iii) exibir o conteúdo do seu primeiro elemento. O código em C apresentado tem um comportamento bem definido ou sua execução poderia ser interrompida devido a um erro de acesso à memória? Caso você considere que este programa tem um ou mais erros, indique as linhas que contêm problemas, explique-os e mostre como eles poderiam ser corrigidos de maneira que o código possa funcionar adequadamente, mantendo seus objetivos e divisão em funções.

```
 1: #include <stdlib.h>
 2: #include <stdio.h>
 3:
 4: typedef struct no {
 5:   int v;
 6:   struct no* prox;
 7: } No;
 8:
 9: void inicia(No** p) {
10:   p = NULL;
11: }
12:
13: void insere_inicio(No** p, int v) {
14:   No* n = (No*) malloc (sizeof (struct no));
15:   n->v = v;
16:   n->prox = *p;
17:   p = &n;
18: }
19:
20: int main() {
21:   No **p;
22:   inicia(p);
23:   insere_inicio(p, 10);
24:   printf ("Valor do primeiro elemento: %d\n", (*p)->v);
25:   return 0;
26: }
```